⇒ Standardizing the notion of Running Time

⇒ Big-O notation; asymptotic analysis

Consideration 1: Running time typically depends on input size

Example: Insertion sort takes longer in larger arrays

Idea: Parametrize running time by input size

$T(n)$ – running time on inputs of size n

Consideration 2: Running time may depend on other features of the input (not only size)

Example: Insertion sort's running time depends on how sorted input list is

Idea: Consider worst-case input.

Consideration 3: May depend on your computer.

Example: Some computers are faster.

Idea: Assume all basic steps take constant time (say 1 unit)

What is a basic step? It depends but we typically assume that they include

⇒ Arithmetic (with "small" numbers)

- addition, substraction, multiplication, mod, etc.

⇒ Comparisons (of small quantities)

⇒ Data movement (small amounts; i.e., a number)

- Load, Store, Copy

Definition (Running Time)

$T(n)$ = number of basic steps on worst-case input of size n.

Example 1:

Input: Positive integer n

```
sum = 0
for i = 1 to n^2
    sum = sum + 1.

m = 1
for i = 1 to n
    m = m . i

return sum + m;
```

Running time?

Number of sums: $2n^2 + n + 3$

Number of variable assignments: $2n^2 + 2n + 4$

Number of multiplications: n

Number of comparisons: $n^2 + n + 2$

So:

$$T(n) = (2n^2+n+3)\cdot C_{sum} + n \cdot C_{mult} + (n^2+n+2)\cdot C_{comp} + (2n^2+2n+4)\cdot C_{store}$$

- C_{sum}: time it takes to sum two numbers
- C_{mult}: time it takes to multiply two numbers
- C_{comp}: time it takes to compare two numbers
- C_{store}: time it takes to assign a variable

Under our assumptions: $C_{sum} = C_{mult} = C_{comp} = C_{store} = 1$ (unit time).

$$T(n) = 2n^2+n+3+n+n^2+n+2+2n^2+2n+4$$

$$T(n) = 5n^2 + 5n + 9 \Rightarrow \text{running time}$$

This is a lot better! than what we did for Insertion Sort, but we would like even simpler expressions for convenience, simpler comparison of algorithms, etc.

⊗ ⇒ We want to report only the term of $T(n)$ that dominates its growth. So we will say that:

$$T(n) = 3n^2 + 3n + 1 = O(n^2)$$

↳ Ignoring lower order terms and all constants!!!

Let's make this formal.

Big O notation:

Definition For non-negative functions f and g, we say that

$$f(n) = O(g(n))$$

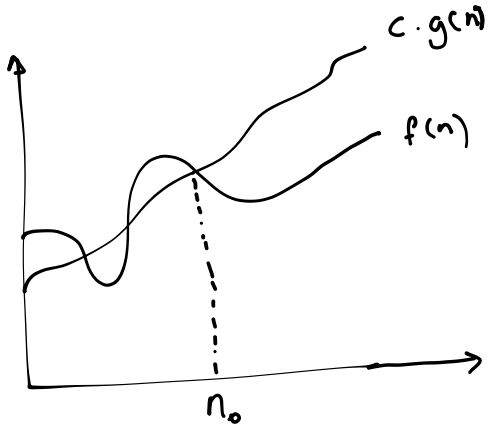
if $\exists$ constants $c > 0$ and $n_0 > 0$ such that

$$f(n) \leq c \cdot g(n)$$

for all $n \geq n_0$

In words: The function $f(n)$ grows at most as fast as $c \cdot g(n)$.

In pictures:



Example 1:

$$6n^2 = O(n^3).$$

Proof Need to find c and n_0 such that $6n^2 \leq cn^3 \quad \forall n \geq n_0$

Take $c = 6$ and $n_0 = 1$.

(This is not the unique combination of c and n_0)

(n_0 and c can depend on each other.)

Example 2:

$$6n^2 = O(n^2)$$

Proof Take $c = 6, n_0 = 1$.

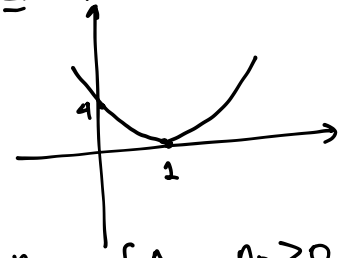
Example 3:

$$4n^2+8n-4 = O(n^2)$$

Proof: Take $C=8$. When does the following holds?

$$4n^2+8n-4 \leq 8\cdot n^2$$

$$0 \leq 4n^2-8n+4 = 4(n-1)^2$$

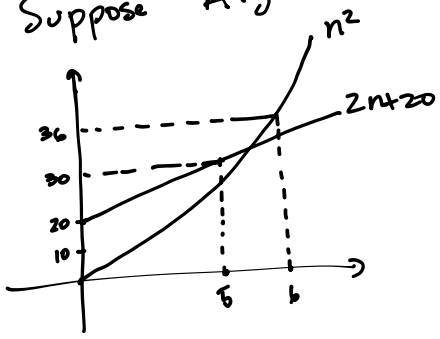


Enough to take $n_0=2$ [Any $n_0>0$ would work] ■

Example 4:

Suppose Algorithm 1 takes $2n+20$ basic steps.

Suppose Algorithm 2 takes n^2 basic steps



- For $n \leq 5$ Algorithm 2 is faster.
- For $n \geq 6$ Algorithm 1 is faster.
- With big O notation we can ignore what happens for small values of n.
- $2n+20 = O(n)$

  $C=3$, $n_0=20$ → $2n+20 \leq 3n$, $20 \leq n$
- $n^2 = O(n^2)$

  $c_0=1$, $n_0=1$

In Conclusion: Algorithm 1 has better asymptotic running time. [i.e., for large input sizes, Algorithm 1 is faster]

⇒ Big O notation is for upper bounds. What about lower bounds?

Definition For non-negative functions f and g, we say that

$$f(n) = \Omega(g(n))$$

If $\exists\ c > 0$ and $n_0 > 0$ such that

$$f(n) \geq c \cdot g(n)$$

$\forall n \geq n_0$

In words: f is bounded from below by g (asymptotically)

Example 1:

$$n^2 = \Omega(n)$$

$c = 1$, $n_0 = 1$

$$n^2 \geq n$$

Example 2

$$n^2 = \Omega(n^2)$$

$c = 1$, $n_0 = 1$

Observation: $f(n)$ can be $O(g(n))$ and $\Omega(g(n))$. In this case, we say:

$$f(n) = \Theta(g(n))$$

↳ "theta"

In words: $f(n)$ and $g(n)$ are of the same order (asymptotically).

Some useful facts

1. Multiplicative constants can be omitted.

   $a \cdot f(n) = O(f(n))$ Proof: Take $c = a$ and $n_0 = 1$.

2. $n^b = O(n^a)$ for $a \geq b$.

Proof: $c = 1, n_0 = 1$

$n^b \leq n^a$ for $a \geq b$. ∎

3 If $f(n) = O(h(n))$ and $g(n) = O(h(n))$. Then,

$f(n) + g(n) = O(h(n))$.

Proof:

$\exists\ c, c', n_0, n_0'$ such that $\forall n \geq \max\{n_0, n_0'\}$

$f(n) \leq c \cdot h(n)$, and

$g(n) \leq c' \cdot h(n)$

Adding up: $f(n) + g(n) \leq (c + c') \cdot h(n)$

4 Polynomials are easy

$a_d \cdot n^d + a_{d-1} \cdot n^{d-1} + \ldots + a_1 \cdot n + a_0 = O(n^d)$

Proof. Follows from 1, 2 and 3.

5 Constants are $O(1)$. Proof: Take c = constant.

6 Any exponential dominates any polynomial.

Example: $n^5 = O(2^n)$

$c = 3 \quad n_0 = 21$

$n^5 \leq 3 \cdot 2^n$

7 Any polynomial dominates any logarithm.

Examples: $(\log n)^3 = O(n) \qquad (\log n)^{100} = O(n)$

$n \log n = O(n^2)$

8 The basis of logarithms are irrelevant.

$$\log_a n = \Theta(\log_b n)$$

* The proofs of 6, 7 and 8 are left as exercises.

Other Key Properties:

Fact 1: Let f and g be two functions such that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

exists and is equal to some number $a > 0$.

Then $f(n) = \Theta(g(n))$.

Proof: From the definition of limit ("(ε,δ)-definition"), we have that for any $\varepsilon > 0$ $\exists\, n_0$ such that $\forall n \geq n_0$

$$\left| \frac{f(n)}{g(n)} - a \right| \leq \varepsilon$$

Then:

$$\left| f(n) - ag(n) \right| \leq \varepsilon \cdot g(n)$$

$$(a-\varepsilon)\cdot g(n) \leq f(n) \leq (a+\varepsilon)\, g(n)$$

Hence, $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

So, $f(n) = \Theta(g(n))$ ∎

Fact 2:

- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq \infty$, then $f = O(g)$
- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$ then $f = \Omega(g)$.

Proof. Exercise.

Note: While limits are very useful for determining O and Ω, there are drawbacks (e.g., sometimes they do not exist).

Fact 3 (Transitivity)

Suppose $f = O(g(n))$ and $g(n) = O(h(n))$.

Then, $f = O(h(n))$.

Proof $\exists\ c > 0, n_0 > 0$ such that for all $n \geq n_0$

$f(n) \leq c\, g(n)$.

$\exists\ c' > 0, n_0' > 0$ such that $\forall n \geq n_0'$

$g(n) \leq c' \cdot h(n)$

So, $f(n) \leq c \cdot c' \cdot h(n)$ $\forall\ n \geq \max\{n_0, n_0'\}$. ∎

Fact 4 If $f(n) = O(h_1(n))$ and $g(n) = O(h_2(n))$.

Then, $f(n) \cdot g(n) = O(h_1(n) \cdot h_2(n))$

Proof: Exercise

Recap True or False?

① $n^2 - 5n - 100 = O(n)$ **False**

If it were true, $\exists\ c, n_0$ s.t. $\forall n \geq n_0$

$n^2 - 5n - 100 \leq cn$

$n^2 - (5+c)n - 100 \leq 0$.

This false for any $c \geq 0$ provided n is large enough.

② $n^2 + O(n) = O(n^2)$ **True**

By convention, $n^2 + O(n)$ means $n^2 + f(n)$ for some $f(n) = O(n)$.

So, it follows from 2 and 3 above.

③ $2^{n+1} = O(2^n)$ **True**

$2^{n+1} = 2 \cdot 2^n = O(2^n)$.

④ $2^{2n} = O(2^n)$ **False.**

$2^{2n} = 4^n$. Then, we would need $c, n_0 : \forall n \geq n_0$

$4^n \leq c \cdot 2^n \Rightarrow 2^n \leq c$ which is false for n large.

⑤ $\log n^2 = O(\log n)$ **True**

$\log n^2 = 2 \log n = O(\log n)$

⑥ $5n - O(n) = \Omega(n)$ **False**

Take $f(n) = 5n - \sqrt{n}$

$5n - f(n) = 5n - O(n)$

But, $5n - f(n) = \sqrt{n}$ which is NOT $\Omega(n)$.

Note: Fastest computers in the world do $\sim 10^{18}$ steps/s.

Suppose $n = 100,000,000$ (input size)

# of operations	vs	time
n		< 1 ms
$n \log_{10} n$		< 1 ms
n^2		~ 10 ms
n^3		~ 12 days
n^4		~ 3 million years
2^n		...